FRANKLIN GRAY PATENTS, LLC
ROBERT H. FRANTZ, REGISTERED US PATENT AGENT

FACSIMILE TRANSMISSION

TO: Examiner Jason D. Mitchell
U.S. Patent and Trademark Office
GAU 2193
Fax: 571-273-8300 (Central Fax Server)

FROM: Robert H. Frantz
Franklin Gray Patents, LLC
Tel: 405-812-5613
Fax: 405-440-2465

DATE: November 7, 2005

PAGES: 19 (inclusive)

In re the Application of:
Lorin Ullmann)
Serial No. **10/047,011**)
Filed: 01/16/2002)
Docket: **AUS920010750US1**)
For: "Stack Unique Signatures for)
Program Procedures and Methods")

Group: 2193
Examiner: Jason D. Mitchell

Certificate of Transmission under 37 CFR §1.8
I hereby certify that this correspondence is being facsimile transmitted to the Patent and Trademark Office on:

DATE: Nov. 7, 2005

SIGNATURE: Robert Frantz

Robert H. Frantz, Reg. No. 42,553

NOTICE:

P.O. Box 23324 Oklahoma City, OK 73123-2334
Tel.: 405-812-5613 Fax.: 405-440-2465
rfrantz@franklingray.com www.franklingray.com

PTO/SB/17 (12-04v2)
Approved for use through 07/31/2006. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Effective on 12/08/2004.
Fees pursuant to the Consolidated Appropriations Act, 2005 (H.R. 4818).

FEE TRANSMITTAL For FY 2005

☐ Applicant claims small entity status. See 37 CFR 1.27

TOTAL AMOUNT OF PAYMENT ($) 500.00

Complete If Known	
Application Number	10/047,011
Filing Date	01/16/2002
First Named Inventor	Lorin Ullmann
Examiner Name	Jason D. Mitchell
Art Unit	2193
Attorney Docket No.	AUS920010750US1

METHOD OF PAYMENT (check all that apply)

☐ Check ☐ Credit Card ☐ Money Order ☐ None ☐ Other (please identify): _____

☑ Deposit Account Deposit Account Number: 09-0447 Deposit Account Name: Int'l Business Machines

For the above-identified deposit account, the Director is hereby authorized to: (check all that apply)

☑ Charge fee(s) indicated below ☐ Charge fee(s) indicated below, except for the filing fee

☑ Charge any additional fee(s) or underpayments of fee(s) under 37 CFR 1.16 and 1.17 ☑ Credit any overpayments

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

FEE CALCULATION

1. BASIC FILING, SEARCH, AND EXAMINATION FEES

Application Type	FILING FEES Fee ($)	FILING FEES Small Entity Fee ($)	SEARCH FEES Fee ($)	SEARCH FEES Small Entity Fee ($)	EXAMINATION FEES Fee ($)	EXAMINATION FEES Small Entity Fee ($)	Fees Paid ($)
Utility	300	150	500	250	200	100	
Design	200	100	100	50	130	65	
Plant	200	100	300	150	160	80	
Reissue	300	150	500	250	600	300	
Provisional	200	100	0	0	0	0	

2. EXCESS CLAIM FEES

Fee Description	Fee ($)	Small Entity Fee ($)
Each claim over 20 (including Reissues)	50	25
Each independent claim over 3 (including Reissues)	200	100
Multiple dependent claims	360	180

Total Claims _____ - 20 or HP = _____ x _____ = _____ Fee Paid ($)

HP = highest number of total claims paid for, if greater than 20.

Indep. Claims _____ - 3 or HP = _____ x _____ = _____ Fee Paid ($)

HP = highest number of independent claims paid for, if greater than 3.

Multiple Dependent Claims Fee ($) _____ Fee Paid ($) _____

3. APPLICATION SIZE FEE

If the specification and drawings exceed 100 sheets of paper (excluding electronically filed sequence or computer listings under 37 CFR 1.52(e)), the application size fee due is $250 ($125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).

Total Sheets _____ - 100 = _____ Extra Sheets / 50 = _____ Number of each additional 50 or fraction thereof (round up to a whole number) x Fee ($) _____ = Fee Paid ($) _____

4. OTHER FEE(S)

Non-English Specification, $130 fee (no small entity discount) Fees Paid ($) _____

Other (e.g., late filing surcharge): Fee for Filing a Brief in Support of an Appeal – 37 CFR 41.20(b)(2) 500

SUBMITTED BY

Signature	Robert Frantz	Registration No. (Attorney/Agent) 42,553	Telephone 405-812-5613
Name (Print/Type)	Robert H. Frantz		Date November 7, 2005

In the United States Patent and Trademark Office

RECEIVED
CENTRAL FAX CENTER
NOV 0 7 2005

In re the Application of:
Lorin Ullmann)
Serial No. 10/047,011)
Filed: 01/16/2002)
Docket: AUS920010750US1)
For: "Stack Unique Signatures for)
Program Procedures and Methods")

Group: 2124
Examiner: Jason T. Mitchell

APPEAL BRIEF

Real Party in Interest per 37 CFR §41.37(c)(1)(i)

The subject patent application is owned by International Business Machines Corporation of Armonk, NY.

Related Appeals and Interferences per 37 CFR §41.37(c)(1)(ii)

The present patent application is related to US Patent Application number 09/497,606, docket number AUS990893US1, which is now issued US Patent 6,550,058.

Status of Claims per 37 CFR §41.37(c)(1)(iii)

Claims 1 - 18 were finally rejected in the Office Action dated 06/06/2005, which was the second Office Action in the prosecution of this application. The rejections of Claims 1 - 18 were appealed on September 6, 2005.

Status of Amendments after Final Rejections per 37 CFR §41.37(c)(1)(iv)

No amendments to the claims have been submitted or entered after final rejections.

Summary of the Claimed Subject Matter per 37 CFR §41.37(c)(1)(v)

The invention provides a system and method for marking a computer or microprocessor's stack with signatures to indicate which portions of the stack were utilized by one or more re-entrant or object-oriented programming software code modules (pg. 8, lines 2 - 13). Addressing such needs in an object-oriented and re-entrant computing environment is especially challenging, as each code module can be duplicated during runtime, and the duplicates or "instances" of the code modules may execute simultaneously or concurrently. As such, simply having each unique code module place an identifier on the stack would not be sufficient, as it would be difficult to know which instance of each code module executed, when it executed, and what portion of the stack was dedicated or used by each instance (pg. 17, lines 13 - 18).

To solve this problem, the invention performs the following steps as claimed in Claims 1 and 7:

(a) inserts "stack signing software" into one or more re-entrant or object-oriented programming code modules which are stored in a computer-readable medium (fig. 4 #45; pg. 15 lines 1 - 3);

(b) produces one or more executable programs containing one or more executable re-entrant or object oriented programming code modules containing said inserted stack signing software (fig. 4 #48, #49, #400; pg. 15 lines 4 - 7); and

(c) upon subsequent execution of said executable re-entrant or object-oriented code modules, assigning unique module identifier values to said code modules by said stack signing software (pg. 17 line 19 - pg. 18 line 13, pg. 19 lines 5 - 8), wherein the stack signing software prevents module identifiers from having a same value for multiple instances of any re-entered or multiply instantiated code module (pg. 17 line 19 - pg. 19 line 2), and pushing onto said processing stack said stack signatures within stack frames allocated to said code modules (pg. 19 lines 5 - 8).

Claim 13 sets forth a system embodiment of the present invention that interoperates with a software compiler, including:

(a) a control operable by a user to indicate whether or not to insert stack signature marking code segments into application software modules (fig. 4 #44, pg. 14 lines 18 - 21; pg. 15 line 11 - pg. 16 line 15);

(b) a code inserter which, responsive to the operation of the control means, searches for entry points and exits points in application software modules and inserts stack signature marking code segments following each entry point and prior to each exit point into said application software modules (fig. 4 #47, pg. 14 line 22);

(c) a compiler for producing one or more executable programs containing one or more executable re-entrant or object oriented programming code modules containing the inserted stack signing software (pg. 15 line 6); and

(d) a software debugger enhanced such that upon execution of the executable re-entrant or object-oriented code modules, unique module identifier values are assigned to the code module instances by the inserted stack signing software, wherein the stack signing software prevents module identifiers from having a same value for multiple instances of any re-entered or multiply instantiated code module, and the stack signatures are pushed onto the processing stack (see disclosure references for the method and computer-readable medium claims above for these steps or operations).

Grounds for Rejection For Which Review is Sought per 37 CFR §41.37(c)(1)(vi)

Review by the Board is requested of the rejections of:

(1) Claims 1 - 3 and 7 - 9 under 35 U.S.C. §103(a) as being unpatentable over U.S. patent 5,950,003 to Kaneshiro, *et al.* (hereinafter "Kaneshiro") in view of U.S. patent 6,807,583 to Hrischuk, *et al.* (hereinafter "Hrischuk");

(2) Claims 4 - 6 and 10 - 15 under 35 U.S.C. §103(a) as being unpatentable over Kaneshiro in view of Hrischuk in further view of U.S. patent 6,161,219 to Ramkumar, *et al.* (hereinafter "Ramkumar");

(3) Claims 16 - 17 under 35 U.S.C. §103(a) as being unpatentable over Kaneshiro in view of Hrischuk, in further view of U.S. patent 6,862,696 to Voas (hereinafter "Voas"); and

(4) Claim 18 under 35 U.S.C. §103(a) as being unpatentable over Kaneshiro in view of Hrischuk in further view of U.S. patent to Carter, *et al.* (hereinafter "Carter")

Arguments per 37 CFR §41.37(c)(1)(vii)

Rejections of Claims 1 - 3 and 7 - 9 under 35 U.S.C. §103(a) over Kaneshiro in view of Hrischuk

Hrischuk is cited in the rationale for the rejections of Claims 1 - 3 and 7 - 9 as the secondary reference in this combination to teach instrumentation of object oriented executable code and instance counters because Kaneshiro does not teach these aspects of our independent claims. In particular, it was argued that Hrischuk teaches assigning an "instance number".

However, Hrischuk mentions *only once* the step or element of assigning an "instance number" as follows:

> A process thread is identified with the process scenario name and the process thread identifier, such as <L,k>. If an object-oriented system is being monitored then the object identifier should include class name and **instance number** of an executing object.
> *(Hrischuk Col. 23, lines 19 - 22, emphasis added)*

A text search of the HTML version of the Hrischuk patent on the USPTO database for the term "instance number" finds only this single occurrence in the entire Hrischuk disclosure of the term "instance number". No further citations were provided in the rationale for the rejections to support the conclusion that Hrischuk teaches the instance number assignment process we have claimed.

As can be seen from this passage, Hrischuk is silent as to *how* their instance number is assigned. For example, is it pre-assigned, or is it a unique value? Is it generated using a pseudo-random generator? Is it generated during runtime, during compilation, or by the software designer himself? Does Hrischuk's invention make sure that no two module identifiers are the same value?

We have claimed that our invention prevents instance identifiers from having the same value (Claims 1 and 7), and in particular that the identifier values are generated using a counting process (Claims 2 and 8). Claims 3 and 9 also include the steps or limitations as claimed in Claims 1 and 7, as they depend from these claims, respectively. Hrischuk is silent as to these teachings, and therefore the rejections of claims 1 - 3 and 7 - 9 should be reversed.

Further, Hrischuk is also silent as to *where* their instance numbers are stored. For example, are they stored on a stack, as we have claimed, or in a data structure, such as a variables or an array of variables? We have claimed "pushing" the instance identifier, with certain other data, onto a processing stack. Hrischuk is silent as to these teachings, and therefore the rejections of claims 1 - 3 and 7 - 9 should be reversed.

Still further, Hrischuk uses a communications protocol, not stack signatures, to record events and data associated with events:

> In order to record sufficient information regarding software process execution, angio tracing is employed. Angio tracing according to the invention identifies scenario and potential precedence relationships between recorded events of an application and properly characterises concurrency. Angio tracing according to the invention characterises **communication protocol elements** in the form of blocking request initiation, non-blocking request initiation, request acceptance, synchronisation acceptance, sending a reply to a blocking request initiation, and acceptance of a reply. Angio tracing supports integration of information from a heterogeneous environment because it is independent of implementation technology, execution environment, and monitoring approach. Optionally, multiple angio traces are recorded simultaneously. Automated trace analysis is possible because angio tracing according to an embodiment is based on a formal model. (Col. 16, lines 38 - 32, emphasis added)
>
> ...
>
> RPC, synchronisation, and asynchronous communication protocols are also characterised by the following elements:
>
> Blocking request initiation: An object cannot proceed until it receives a reply to a request it has just made;

> Non-blocking request initiation: An initiating or forwarding object makes a service request to another object and the initiating object does not block to wait for a reply;
>
> Request acceptance: A blocked responding object accepts a new service request and begins a new period;
>
> Synchronisation acceptance: A responding object is already processing a service request but it is blocked, waiting to accept another message to continue the service;
>
> Sending a reply to a blocking request: A replier object sends the reply to the blocked initiating object; and,
>
> Acceptance of a reply: A blocked initiating object receives the reply and continues execution. (Col. 21, lines 6 - 23)

A text search the HTML version of the Hrischuk patent on the USPTO database for the terms "stack" and "heap", where "heap" is a term used sometimes in place of "stack", finds no occurrences of the terms "stack" or "heap" in the entire disclosure.

For these reasons, Kaneshiro in view of Hrischuk fails to teach generating an instance value according to our process of assigning instance values, and storing instance values in signature blocks in a computer stack. Reversal of the rejections of Claims 1 - 3 and 7 - 9 is requested.

Rejections of Claims 4 - 6 and 10 - 12 under 35 U.S.C. §103(a) over Kaneshiro in view of Hrischuk in further view of Ramkumar

Claims 4 - 6 depend from Claim 1, and Claims 10 - 12 depend from Claim 7. According to the rationale for the rejections of these claims, Ramkumar is not employed to teach the missing steps or limitations as discussed in the foregoing paragraphs regarding the rejections of claims 1 and 7. For these same reasons, reversal of the rejections of claims 4 - 6 and 10 - 12 is requested.

Rejections of Claims 13 - 15 under 35 U.S.C. §103(a) over Kaneshiro in view of Hrischuk in further view of Ramkumar

Claim 13 is an independent system claim, and Claims 14 and 15 depend from Claim 13. In particular, we have claimed in Claim 13 a compiler means for inserting stack signing

software, wherein the stack signing software assigns identifier values in a manner that prevents identifiers of instances from having the same value, as previously discussed with regard to the rejections of Claims 1 and 7. Claims 14 and 15 include these same elements or limitations. According to the rationale for the rejections of these claims, Ramkumar is not employed to teach the missing steps or limitations as discussed in the foregoing paragraphs regarding the rejections of claims 1 and 7. For these same reasons, reversal of the rejections of claims 13 - 15.

Rejections of Claims 16 - 17 under 35 U.S.C. §103(a) over Kaneshiro in view of Hrischuk in further view of Voas

In the rationale for the rejections of Claims 16 - 17, Voas was employed to teach our claimed step, element or limitation of encryption of the stack signature.

Voas discloses encryption of "test data" in order to transfer collected data from an application program being used "in the field" (e.g. on a client computer) to a certification laboratory (their "SCL"). This is done to protect the data in transit as it is being transmitted from one computer to another, and not to protect data stored on a computer stack.

Voas is silent as to encrypting signatures which are stored on a computer or processing stack as we have claimed. A "computer stack" is not part of a transmission protocol between two computers, but instead is an internal operational component of the computer's processor and/or operating system. While the term "protocol stack" often is used to refer to a communication protocol which is divided into "layers" of functionality, the term "processing stack" is not synonymous with the term "protocol stack". Further, the term "push" is not used with a *protocol* stack, but is commonly used when referring to moving data into or onto a *processing* stack.

For example, Random House Webster's Computer and Internet Dictionary, Third Edition, (copyright 1999), defines a stack as follows:

stack

1. In programming, a special type of data structure in which items are removed in the reverse order from that in which they are added, so the most recently added item is the first one removed. This is also called *last-in, first-out (LIFO)*. Adding an item to a stack is called *pushing*. Removing an item from a stack is called

popping.

2. In networking, short for *protocol stack.*

3. In Apple Computer's HyperCard software system, a stack is a collection of cards.

For these reasons, neither Kaneshiro, Hrischuk, or Voas teach encryption of stack signatures as we have claimed, wherein the identifier information is encrypted and stored on a processing stack. Reversal of the rejections of Claims 16 - 17 is requested.

Claim 18 under 35 U.S.C. §103(a) over Kaneshiro in view of Hrischuk in further view of Carter

Claim 18 claims generation of unique instance identifiers to be stored on a processing stack as previously discussed relative to Claim 1, as Claim 18 depends from Claim 1. Thus, the rejections of Claim 18 fail to teach the steps and limitations as set forth in the foregoing arguments requesting reversal of the rejection of Claim 1.

Claim 18 also sets forth a limitation regarding the method of generation of the instance identifier using a pseudo-random number generator, instead of or in cooperation with a counter. This embodiment helps assure instance number uniqueness without complicated inter-process communications to check out instance numbers. In the rationale for the rejection of Claim 18, it was reasoned that Carter teaches this step or limitation.

Carter is directed towards "building" or "compiling" new versions of programs which are backwards compatible with older versions of the same program, and especially towards building a new version "object servers" which can be utilized by older programs which interfaced to and used the previous version of the object server.

The examiner has cited the following as teaching our claimed process of generation of instance numbers using pseudo-random number generation:

> When automatic version compatible object server **building** is not selected (such as when **compiling** an initial version of an object server), **compiler** 52 generates a new vtable offsets and **identifiers** in steps 140-150. The resulting object server 92 therefore is extremely unlikely to support the same interfaces as any other object server. (The bit-length of

> the library ID, CLSIDs, and IIDs, and **OLE 2's random generation routine used in the preferred embodiment to generate these identifiers, guarantees to a very high probability that no two identifiers generated by compiler 52 are identical.**) When automatic version compatible object server building is selected, however, compiler 52 first performs process 76 (FIG. 6) to determine whether new object server 64 (FIG. 2) can be made version compatible with existing object server 66 (FIG. 2). Then, in process 78 (FIG. 5), compiler assigns vtable offsets and identifiers of new object server according to the new object server's compatibility as described more fully below. (Col. 12, lines 44 - 61, emphasis added)

Clearly, Carter is generating their identifiers *during compilation*, not during runtime or during execution of the code to be monitored. Additionally, the generation of Carter's identifiers is being performed *by a compiler*, not by the code inserted into the actual application program.

In Claim 18 (and Claim 1), we have specified that our instance numbers are generated "upon execution of said executable re-entrant or object-oriented code modules". Carter is silent as to such runtime generation of instance numbers. Further, as it would be impossible at the time of source code compilation to know how many instances of a particular object would be simultaneously instantiated during later runtime, it would be impossible to generate our instance numbers during compilation or build of a program.

For these reasons, Kaneshiro in view of Hrischuk in further view of Carter does not teach all of our claimed elements, steps, or limitations as set forth in Claim 18. Reversal of the rejection of Claim 18 is requested.

Respectfully Submitted,

Robert Frantz

Agent for Appellant(s)
Robert H. Frantz, Reg. No. 42,553
Tel: (405) 812-5613

Franklin Gray Patents, LLC
P.O. Box 23324
Oklahoma City, OK 73127
Tel: 405-812-5613
Fax: 405-440-2465

Claims Appendix

per 37 CFR §41.37(c)(1)(viii)

Clean Form of Amended Claims

Claim 1 (previously amended):

A method for marking a processing stack with signatures to indicate which portions of the stack were utilized by one or more re-entrant or object-oriented programming software code modules, said method comprising the steps of:

inserting stack signing software into one or more re-entrant or object-oriented programming code modules stored in a computer-readable medium;

producing one or more executable programs containing one or more executable re-entrant or object oriented programming code modules containing said inserted stack signing software;

upon execution of said executable re-entrant or object-oriented code modules, assigning unique module identifier values to said code modules by said stack signing software, said stack signing software preventing module identifiers from having a same value for multiple instances of any re-entered or multiply instantiated code module, and pushing onto said processing stack said stack signatures within stack frames allocated to said code modules.

Claim 2 (previously amended):

The method as set forth in Claim 1 further comprising the steps of:

generating an instance count for each instantiation of executable code module in said stack signature for each object instance dynamically created during runtime of a re-entrant executable code module; and

pushing onto said processing stack said instance count.

Claim 3 (original):

The method as set forth in Claim 1 further comprising the step of pushing onto said stack an entry/exit indicator associated with said unique module identifier.

Claim 4 (original):

The method as set forth in Claim 1 further comprising the step of inserting stack signature marking software segments into application source code, said insertion step being performed prior to compilation of said application source code.

Claim 5 (original):

The method as set forth in Claim 4 further comprising the step of providing a global control which indicates all application source code modules are to have stack signature marking software segments inserted into them during a given compilation job.

Claim 6 (original):

The method as set forth in Claim 4 further comprising the step of providing a selective control which indicates only certain application source code modules are to have stack signature marking software segments inserted into them during a given compilation job.

Claim 7 (previously amended):

A computer readable medium encoded with software for marking a processing stack with signatures to indicate which portions of the stack were utilized by which application code modules, said software causing a processor to perform the steps of:

inserting stack signing software into one or more re-entrant or object-oriented programming code modules stored in a computer-readable medium;

producing one or more executable programs containing one or more executable re-entrant or object oriented programming code modules containing said inserted stack signing software;

upon execution of said executable re-entrant or object-oriented code modules, assigning unique module identifier values to said code modules by said stack signing software, said stack signing software preventing module identifiers from having a same value for multiple instances of any re-entered or multiply instantiated code module, and pushing onto said processing stack said stack signatures within stack frames allocated to said code modules.

Claim 8 (original):

The computer readable medium as set forth in Claim 7 further comprising software to perform the steps of:

generating an instance number for each instantiation of a code module; and

pushing onto said processing stack said instance numbers associated with said unique module identifier values.

Claim 9 (original):

The computer readable medium as set forth in Claim 7 further comprising software for performing the step of pushing onto said stack an entry/exit indicator associated with said unique module identifier.

Claim 10 (original):

The computer readable medium as set forth in Claim 7 further comprising software for performing the step of inserting stack signature marking software segments into application source code, said insertion step being performed prior to compilation of said application source code.

Claim 11 (original):

The computer readable medium as set forth in Claim 10 further comprising software for performing the step of providing a global control which indicates all application source code modules are to have stack signature marking software segments inserted into them during a given compilation job.

Claim 12 (original):

The computer readable medium as set forth in Claim 10 further comprising software for performing the step of providing a selective control which indicates only certain application source code modules are to have stack signature marking software segments inserted into them during a given compilation job.

Claim 13 (previously amended):

A system for inserting stack signature marking code segments into application software modules prior to compilation, said system cooperating with a compiler and comprising:

a control means operable by a user to indicate whether or not to insert stack signature marking code segments into application software modules;

a code insertion means which, responsive to the operation of the control means, searches for entry points and exits points in application software modules and inserts stack signature marking code segments following each entry point and prior to each exit point into said application software modules;

a compiler means for producing one or more executable programs containing one or more executable re-entrant or object oriented programming code modules containing said inserted stack signing software; and

a debugger means configured to, upon execution of said executable re-entrant or object-oriented code modules, assign unique module identifier values to said code modules by said stack signing software, said stack signing software preventing module identifiers from having a same value for multiple instances of any re-entered or multiply instantiated code module, and to push onto said processing stack said stack signatures.

Claim 14 (previously amended):

The system of Claim 13 wherein said control means comprises a global control means for indicating insertion of stack signature marking code segments are to be inserted into all application software modules to be compiled.

Claim 15 (previously amended):

The system of Claim 13 wherein said control means comprises a selective control means for indicating specific applications software modules or groups of application software modules into which stack signature marking code segments are to be inserted.

Claim 16 (previously added):

The method as set forth in Claim 1 further comprising encrypting at least a portion of said stack signature.

Claim 17 (previously added):

The method as set forth in Claim 2 further comprising encrypting said instance number in said stack signature.

Claim 18 (previously added):

The method as set forth in Claim 1 wherein said step of pushing stack signatures onto said processing stack comprises:

generating a pseudo-random identifier for each object instance dynamically created during runtime of a re-entrant executable code module; and

including said pseudo-random identifier in said stack signature pushed onto said processing stack.

Evidence Appendix

per 37 CFR §41.37(c)(1)(ix)

No evidence has been submitted by applicant or examiner pursuant to 37 CFR §§1.130, 1.131, or 1.132.

Related Proceedings Appendix

per 37 CFR §41.37(c)(1)(x)

No decisions have been rendered by a court or the Board in the related proceedings as identified under 37 CFR §41.37(c)(1)(ii).